Architecture, interface, and benchmark specification

GRANT AGREEMENT NUMBER: 101092877

.

Project acronym:	**SYCLOPS**
Project full title:	**Scaling extreme analYtics with Cross architecture acceLeration based on OPen Standards**
Call identifier:	**HORIZON-CL4-2022-DATA-01-05**
Type of action:	**RIA**
Start date:	**01/01/2023**
End date:	**31/12/2025**
Grant agreement no:	**101092877**

D2.1 – Architecture, interface, and benchmark specification

Executive Summary:	D2.1 provides the overall architectural blueprint of SYCLOPS together with inter and intra-layer interface definitions		
WP:	WP2: Architecture		
Author(s):	Suresh Patoria, Fred Buining, Raja Appuswamy, Maurizio Filippone, Pietro Michiardi		
Leading Partner:	HIRO		
Participating Partners:	All partners		
Version:	1.0	**Status:**	Draft
Deliverable Type:	R – document	**Dissemination Level:**	PU – Public
Official Submission Date:	30-06-2023	**Actual Submission Date:**	05-08-2023

Disclaimer

The SYCLOPS consortium consists of the following partners:

No.	Partner Organization Full Name	Partner Organization Short Name	Country
1	EURECOM GIE	EUR	FR
2	INESC ID - INSTITUTO DE ENGENHARIADE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC	PT
3	RUPRECHT-KARLS-UNIVERSITAET HEIDELBERG	UHEI	DE
4	ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE	CERN	CH
5	HIRO MICRODATACENTERS B.V.	HIRO	NL
6	ACCELOM	ACC	FR
7	CODASIP S R O	CSIP	CZ
8	CODEPLAY SOFTWARE LIMITED	CPLAY	UK

Document Revision History

Version	Description	Contributions
0.1	May 23, 2023 - Architecture Structure	HIRO
0.2	Component Description	All Partners
0.3	June 16, 2023 - First draft	HIRO
0.4	July 24, 2023 - Second draft	EURECOM
1.0	August 5, 2023 - Final draft	EURECOM

Authors

Author	Partner
Suresh Patoria	HIRO
Fred Buining	HIRO
Raja Appuswamy	EURECOM
Maurizio Filippone	EURECOM
Pietro Michiardi	EURECOM

Reviewers

Name	Organisation
Aleksandar Ilic	INESC
Vincent Heuveline	UHEI
Axel Naumann	CERN
Nimisha Chaturvedi	ACC
Pavel Zaykov	CSIP
Mehdi Goli	CPLAY

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of Contents

Executive Summary

The main aim of this deliverable (D2.1. Architecture, Interface, and Benchmark Specification) is to provide the overall architectural blueprint of the SYCLOPS hardware--software stack together with inter and intra-layer interface definitions. Given that SYCLOPS is a multi-layered stack, this deliverable ensures that the cross-layer interfaces and APIs are well defined to ensure compatibility. This deliverable also provides a clear specification of each of the three SYCLOPS uses cases, and identifies key metrics that are relevant to measuring the improvement that SYCLOPS brings about for each use case.

The first part of this deliverable (Sections 1 and 2) provides an introduction to the SYCLOPS project and a high-level overview of the overall architecture. The second part of this deliverable (Sections 3, 4) elaborates on the two core layers of the SYCLOPS stack (the Infrastructure and Platform layers) by (i) describing the key components in each layer, and (ii) the interaction and interfaces of these components with each other and with those in the other layer. Finally, the third part of the deliverable (Section 5) provides a detailed description of each use case together with relevant benchmarking methodology.

1 Introduction

The growing popularity of AI and analytics in virtually all application domains has led to a rapid increase in the amount of data gathered by enterprises and scientific institutions alike, with the global datasphere being projected to reach 250ZB by the year 2025. There is significant potential for advancing data analytics at the intersection of many scientific, technology and societal fields (like particle physics and precision oncology covered in this proposal). However, in order to realize these advances in Europe, it is mandatory for the EU to be fully autonomous in processing and analyzing extreme amounts of data. Such an autonomy, in turn, requires technologies, tools, and solutions that can scale in tandem with both the rate of data growth and the computational complexity of analytics and machine learning algorithms.

Modern analytics and AI workloads are incredibly diverse and consist of a range of scalar, vector, and matrix computations. Thus, the traditional workhorse of the computing industry, the general-purpose CPU, cannot be optimized to meet the diverse requirements of such heterogeneous workloads. This shortcoming of CPUs, in combination with the cessation of Dennard scaling, has led to a Cambrian explosion in the adoption of hardware acceleration solutions that can meet the demands of extreme scale AI and analytics workloads. Unfortunately, most popular solutions today, be it vertically-integrated, application-specific accelerators like Google TPU, or generalized, data-parallel SIMT processors like NVIDIA GPU, use proprietary, closed hardware—software stacks as shown in Figure 1, leading to a monopolization of the AI acceleration market by a few large industry players, or individual silos amongst other niche players that use their own proprietary solutions . There are a few open solutions, like OpenMP, but they are primarily catered for HPC applications.

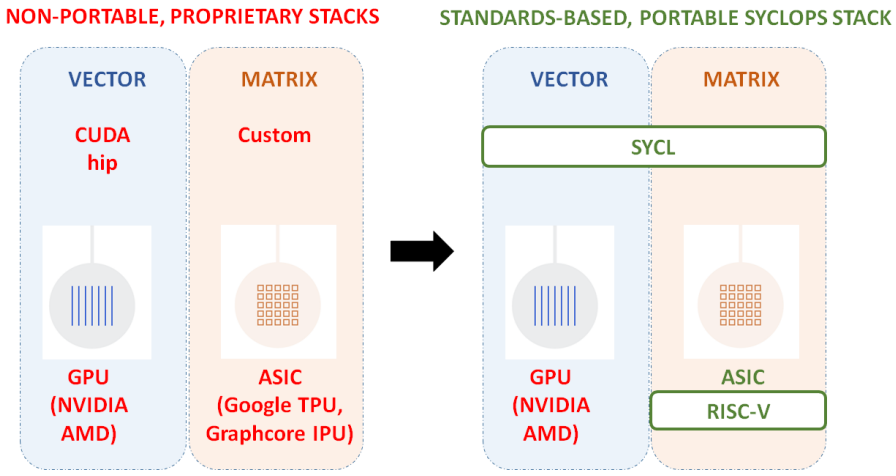

Fig 1. SYCLOPS vision: From closed-source, proprietary AI acceleration solutions (left column) to an open ecosystem of standards-based, interoperable solutions (right column).

The vision of SYCLOPS project is to enable better solutions for AI/data mining for extremely large and diverse data by democratizing AI acceleration using open standards, and enabling a healthy, competitive, innovation-driven ecosystem for Europe and beyond. In order to achieve this vision, SYCLOPS will integrate expertise in computer architecture, programming languages, systems and runtimes, Big Data, High-Performance Computing, autonomous systems, High-Energy Physics, and precision oncology, with the aim of developing novel infrastructure, platform, and application tools for AI acceleration. As shown in Figure 1, this vision relies on the convergence of two important trends in the industry: (i) the standardization and adoption of RISC-V, a free, open Instruction Set Architecture (ISA), for AI and analytics

acceleration, and (ii) the emergence and growth of SYCL as a cross-vendor, cross-architecture, data parallel programming model for all types of accelerators, including RISC-V.

The goal of project SYCLOPS is to bring together these standards for the first time in order to (i) demonstrate ground-breaking advances in performance and scalability of extreme data analytics using a standards-based, fully-open, AI acceleration approach, and (ii) enable the development of inter-operable (open and vendor neutral interfaces/APIs), trustworthy (verifiable and standards-based hardware/software), and green (via application-specific processor customization) AI systems. In doing so, we will use the experience gained in SYCLOPS to contribute back to SYCL and RISC-V standards and foster links to respective academic, industrial and innovator communities (RISC-V foundation, EPI, Khronos, ISO C++). Bringing together the two standards enables codesign in both standards, which in turn, will enable a broader AI accelerator design space, and a richer ecosystem of solutions compared to current proprietary, closed solutions that force a design choice.

In this document, we first provide an overview of the overall architecture of the SYCLOPS hardware—software stack and introduce various abstraction layers (Section 2). Following this, we provide an in-depth discussion of the two core layers of the SYCLOPS stack, namely, infrastructure layer and platform layer, focusing on various components in each layer (Sections 3, 4). Finally, we provide a detailed description of each use case, discuss the libraries that will be implemented in SYCLOPS to support those use cases, and provide an overview of relevant benchmarking methodologies (Section 5).

2 Overall Architecture

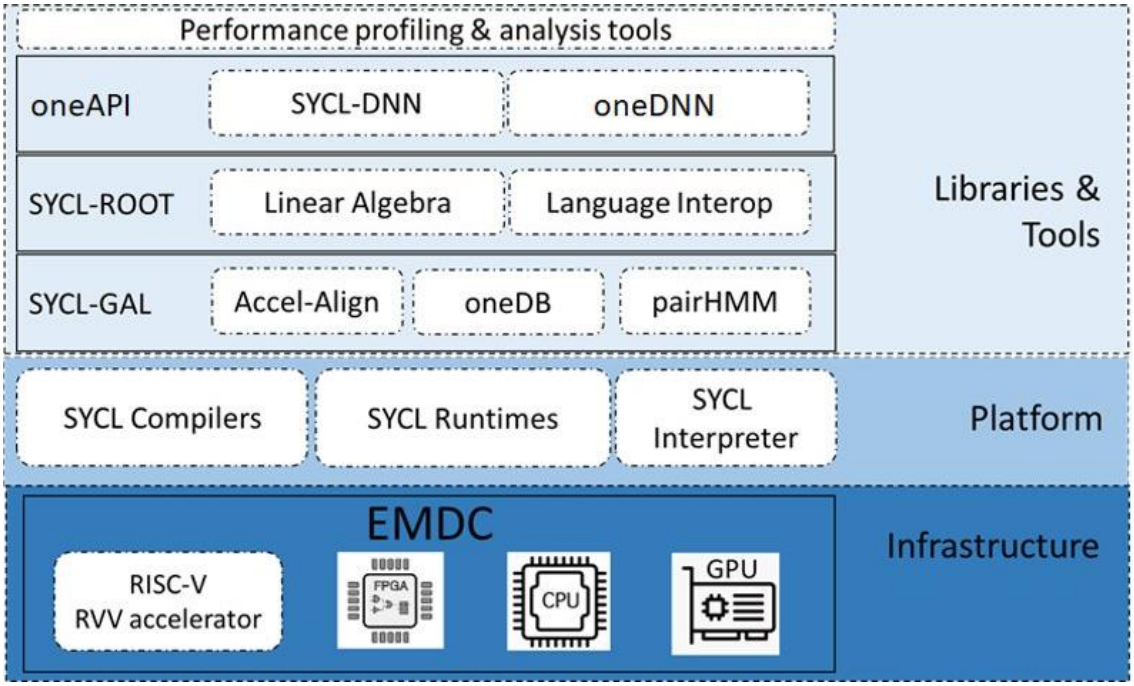

Fig 2. SYCLOPS architecture

The core SYCLOPS hardware—software stack consists of three layers: (i) infrastructure layer, (ii) platform layer, and (iii) application libraries and tools layer.

Infrastructure layer: The SYCLOPS infrastructure layer is the bottom-most layer of the stack and provides heterogeneous hardware with a wide range of accelerators from several vendors. A key accelerator in this layer will be the RISC-V processor designed by CSIP. CSIP will advance their processor description language codAL and their EDA tool Codasip Studio to offer native support for design, verification and implementation of RISC-V processors with customizable vector units. Using these tools, CSIP will develop an RISC-V Vector (RVV) accelerator. In order to demonstrate (i) an end-to-end integration of open standards, and (ii) the cross-architecture, cross-vendor performance portability of SYCLOPS, our partner HIRO will package the RVV accelerator with CPU, GPU, and FPGA from several other leading processor manufactures (Intel, ARM, NVIDIA, Xilinx) and build modular, energy-efficient edge microdatacenters (EMDC). A key aspect of EMDC design will be investigating the use of state-of-the-art PCI 5.0 for interconnecting processors and high-performance storage. The EMDC will be used as a research and development testbed. It will be hosted by the coordinator (EUR), who will provide shared access to all partners.

Platform layer: The second layer from the bottom, the platform layer, provides the software required to compile, execute, and interpret SYCL applications over processors in the infrastructure layer. SYCLOPS will contain oneAPI DPC++ compiler from CPLAY, and hipSYCL, an open-source SYCL compiler toolchain from UHEI. Both DPC++ and hipSYCL compilers already support a wide range of processor backends for SYCL. In SYCLOPS, they will be extended to support a RISC-V backend with native support for vector instrinsics. In terms of SYCL runtime, the DPC++ runtime will be extended to enable the autonomous systems use case by supporting end-to-end platform software required for offloading computation to the RVV accelerator. hipSYCL, in contrast, will be used as a research vehicle to investigate the implementation of advanced, application-aware, graph-based runtime, and

multi-device work scheduling algorithms. In terms of SYCL interpreters, SYCLOPS will contain Cling from CERN. As mentioned, Cling is a state-of-the-art C++ interpreter that is being used as an interactive code development environment for exploratory analysis. Cling will be extended to natively support SYCL and enable Jupyter notebook-based, accelerated ad-hoc analytics.

Application libraries and tools layer: While the platform layer described above enables direct programming in SYCL, the libraries layer enables API-based programming by providing pre-designed, tuned libraries for various deep learning methods for the PointNet autonomous systems use case (SYCL-DNN), mathematical operators for scalable HEP analysis (SYCL-ROOT), and data parallel algorithms for scalable genomic analysis (SYCL-GAL).

3 Infrastructure Layer

In this section, we describe various components of the infrastructure layer in detail. More specifically, we describe (i) the challenges in developing custom RISC-V accelerators and the suite of solutions from CSIP that will be used to address them, and (ii) the challenges in designing energy-efficient data centers and how the edge microdatacenter design from HIRO will address them in the context of SYCLOPS.

3.1 RISC-V cores and vector accelerator (CSIP)

3.1.1 Rise of RISC-V

There are a large number of microprocessors available in the ASIC market. ARM is the most popular architecture and has the largest market share. ARM has an extensive portfolio of high-quality products spanning essentially the entire processor performance range. Their offering has served the market well in the last nearly four decades with the following licensing models: (i) License an existing microprocessor of the portfolio which cannot be modified in any way (neither the ISA nor the microarchitecture), and (ii) granting the right to implement a customized microarchitecture to the licensee who must build the entire processor, but not change the ISA.

Over the past few years, RISC-V, a standardized, free, open Instruction Set Architecture, has emerged as an alternate solution in the AI acceleration space. At its core, RISC-V has a very simple ISA with 47 instructions. But application verticals can customize the ISA with optional extensions that are often realized through specific licensing deals between a RISC-V vendor and the licensee. Thus, RISC-V creates a new business model that allows companies to provide commercial support and other deliverables that are needed for the processor verification or integration into a chip as shown in Figure 3 (right column), in contrast to the classic ARM-style licensing (left column).

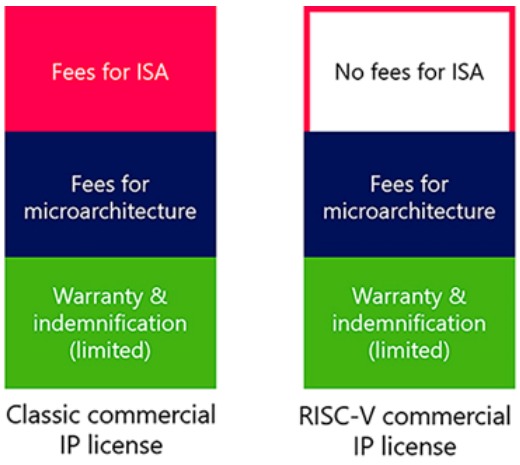

Figure 3. A comparison of business model for state of the art (left hand side, classic commercial) with the potential RISC-V commercial (middle) and fully open source (right) options

However, the design automation tools used by RISC-V vendors today have a major limitation that the implementation and verification of custom extensions is largely a manual process with limited automation. In order to develop a RISC-V-based acceleration solution, one

needs a Hardware development kit (HDK), which is a set of tools necessary to simulate, debug, implement, and verify a RISC-V processor, and a Software development kit (SDK), which is the set of tools necessary to create, simulate, and debug programs for the newly designed processor. As the SDK and HDK are tightly coupled with the processor ISA, each customized RISC-V processor needs its own SDK and HDK. Thus, in order to enable a rapid adoption of RISC-V in the AI acceleration space, the SDK and HDK tools should be ideally autogenerated based on a hardware developer's processor specification. However, current RISC-V vendors do not provide such automation. For instance, the most popular RISC-V Semiconductor Intellectual Property (SIP) providers today are SiFive, based in the US and Andes Technology, based in Taiwan. SiFive offers a range of embedded and application RISC-V processors. They are configurable and have the ability to be customized through their SCIE (SiFive Custom Instruction Extensions). However, using SCIE is largely a manual approach with verification of custom extensions being non-trivial. There is no automation for SDK, and HDK tooling is enhanced manually. Andes Technology has a range of RISC-V cores from embedded (lower-end) to application class (higher-end). Andes CoPilot tool can partially automate the implementation and verification of custom instructions, but the user is left to manually write the Register Transfer Level (RTL) code to implement the new instructions. There is also no automation for SDK and their tools also need to be enhanced manually. This lack of automation creates barriers for hardware developers and hinders RISC-V adoption for AI acceleration.

3.1.2 CSIP background IP

The basis of the CSIP's contribution for the SYCLOPS project comes from two key pillars of established intellectual property:

EDA tool suite: A comprehensive suite of tools that enables the development of custom RISC-V cores and SDKs based on a high-level description in the CodAL language. Our EDA suite comes in two variations - Codasip Studio and CodeSpace. The Codasip Studio also offers the ability to create a virtual prototyping platform, which will expedite the design and development process. Key features of Codasip Studio include:

- **Automated Generation of Software Development Kits (SDKs)**: automatic generation of SDK tailored for each variant of the core design. This SDK includes a compiler, linker, debugger, simulator, and more. The ability to quickly generate this tailored SDK helps to streamline the development process.
- **Virtual Platform Prototyping**: Codasip Studio supports the creation of a virtual platform prototype. This feature provides designers with an environment for early software development and debugging before the physical chip is manufactured.
- **Comprehensive Verification:** The tool suite enables comprehensive verification at all levels - from an Instruction-Accurate (IA) C++ model to RTL - ensuring the highest level of confidence in the core design.
- **Design Space Exploration:** With the Codasip Studio, designers can explore the design space by modifying parameters and assessing the impact on Performance, Power, and Area (PPA).

A lightweight variation of the Codasip Studio is referred to as **Codasip Codespace**. The Codasip CodeSpace package is aimed at the development of software applications. The package contains a graphical user interface (GUI) with perspectives for profiling and debugging and allows easy integration of the Codasip SDK.

Codasip RISC-V cores: A collection of off-the-shelf RISC-V cores suitable for a wide range of use cases - from low-power embedded systems to high-performance application class processors. The RISC-V processor cores come with configurable sets of RISC-V extensions and with configurable interfaces. The software development for the RISC-V cores can be done by either leveraging the Virtual Platform Prototyping capabilities of the Codasip EDA tools or by using a reference design prebuilt for a selected FPGA platform.

3.1.3 CSIP Foreground IP

RISC-V ISA and particularly the newly introduced RISC-V vector extensions (RVV) have started gaining momentum in the AI acceleration space. RVV extensions are a set of instructions that allow RISC-V processors to process data effectively in a Single-Instruction Multiple Data (SIMD) fashion. CSIP contributions in the SYCLOPS project are envisioned in the following three categories:

1. **CodAL improvements:** as our first contribution, we aim to define enhancements/ extensions to the CSIP's processor description language (CodAL) and Codasip Studio APIs extensions to enable simple, cost-effective, and application-driven customization of RVV accelerators.
2. **PPA improvements** as our second contribution, we target integrating CodAL extensions in Codasip Studio, and use it to develop fully customized, RVV acceleration cores that are customized to target the AI and analytics workloads defined by our use cases. The RVV development itself will be carried out outside of the SYCLOPS project. Based on the feedback from the SYCLOPS project, CSIP will aim to prioritize performance, power, and area (PPA) improvements on selected features of the RVV-accelerator. These features could include, but are not limited to the ratified ISA, custom instructions, micro-architecture, and memory subsystem.
3. Contributing to the integration of an RVV reference platform to several other multi-vendor CPUs and GPUs (for comparative evaluation) in an edge microdatacenter (EMDC).

3.1.4 Interfaces & Integration

The RISC-V RVV accelerator will be provided via one or both types of interfaces referred to as following:

- **SW Interface**: A virtual prototyping platform in Codasip Codaspace/Studio for compilation, code generation, and evaluation of code for RISC-V architectures. This will be used for making RISC-V compatible SYCL compilers and for building user applications and analyzing their performance.
- **HW interface**: A precompiled bitstream for a selected FPGA board, where the exact specification depends on the core customizations and necessary interfaces for the remaining Infrastructure components.

Two variations of either the virtual prototyping platform or the precompiled bitstream will be developed in the project.

- **v1.0 (M18)**: An application core based on CSIP application class processors. It is capable of running a full-featured RTOS and offers high levels of customization, from adjusting cache sizes to modifying the number of cores.

- **v2.0 (M33)**: It will be based on the same model but it will include at least partial support of the RISC-V vector extension or some other improvements. Note that the complete implementation of the vector extension is beyond the scope of the SYCLOPS project.

3.2 Edge Micro DataCenter (HIRO)

3.2.1 HIRO EMDC Introduction

Edge computing is essential to realizing the full potential of artificial intelligence (AI), machine learning and internet of things (IoT) as it can dramatically boost services and applications by supporting AI natively, instead of relying on AI in the cloud. The three SYCLOPS application use cases described in Section 5 of this document would all benefit from a powerful edge computing infrastructure that can process large amounts of data at its site of generation without incurring the overhead of data movement to the cloud. Such an infrastructure needs to be highly scalable, compact, and energy efficient. As different workloads benefit from different processing solutions, the edge infrastructure should be capable of modularly integrating any type and quantity of CPUs, GPUs, FPGAs. This is particularly relevant for SYCLOPS whose goal is to demonstrate the portability of the hardware—software stack across multi-vendor hardware.

HIRO will develop an innovative high-performance, reliable edge infrastructure that will be used as a development testbed in SYCLOPS. The EMDC based edge infrastructure is a distributed infrastructure of turn-key datacenters that are fully self-contained (power, cooling, security, etc.) and do not require a dedicated support infrastructure. The total cost of ownership (TCO) of an Edge infrastructure based on a mesh of EMDC's compared to regular DC infrastructure, will be lower, considering the whole infrastructure life cycle (purchase, operation, maintenance, and refresh cycles) and the modifications required in the operating environment (power and cooling infrastructure, building, etc.). There is very little transparency on true data center costs and many studies and calculation models found online, show vendor bias. We have taken an end-to-end power transformation of a regular datacenter and compare it with the energy savings of am EMDC based infrastructure (See Figure 4). The energy savings of the EMDC are created through gravity driven 2 phase cooling, no lumination, no fans, shared redundant and 48V DC power, cognitive engine driven workload and node management.

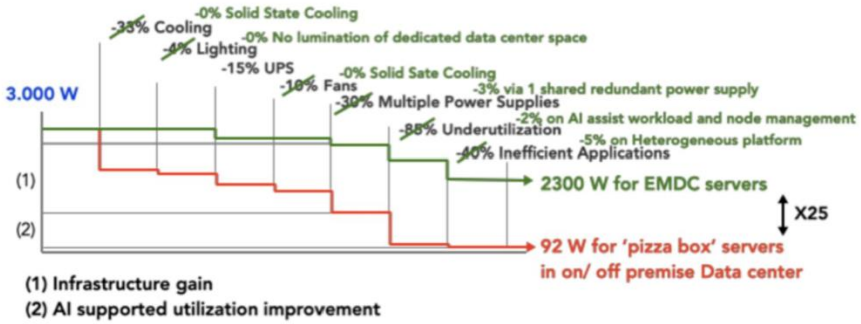

Figure 4. EMDC energy efficiency vs traditional DC energy efficiency

The EMDC infrastructure is built from has power supply, cooling, dual fabric (Ethernet and PCIe switching) and a customizable mix of CPU, GPU, FPGA, NVMe storage. The EMDC

capacities range from 8, 16, 24 nodes in a single enclosure (wall or rackmount) to multiples of 24 nodes (3U) mounted in a rack(s). The cooling is based on a refrigerant and has a PUE of 1.03. At the highest utilization levels of the EMDC, the condenser will require little fan support, at low and medium utilization levels the EMDC operates completely passive. The average PUE currently has stalled around 1.5 with state of the art in Google datacenters of 1.2 and average PUE of 2.0 in small scale edge data centers and server rooms.

The cooling of the EMDC, uses a passive two-phase cooling system to create a new innovative and efficient cooling solution. The cooling loop can handle non-uniformity in the loop (different heat load for each nodes) while being a self-regulated (gravity driven) system.

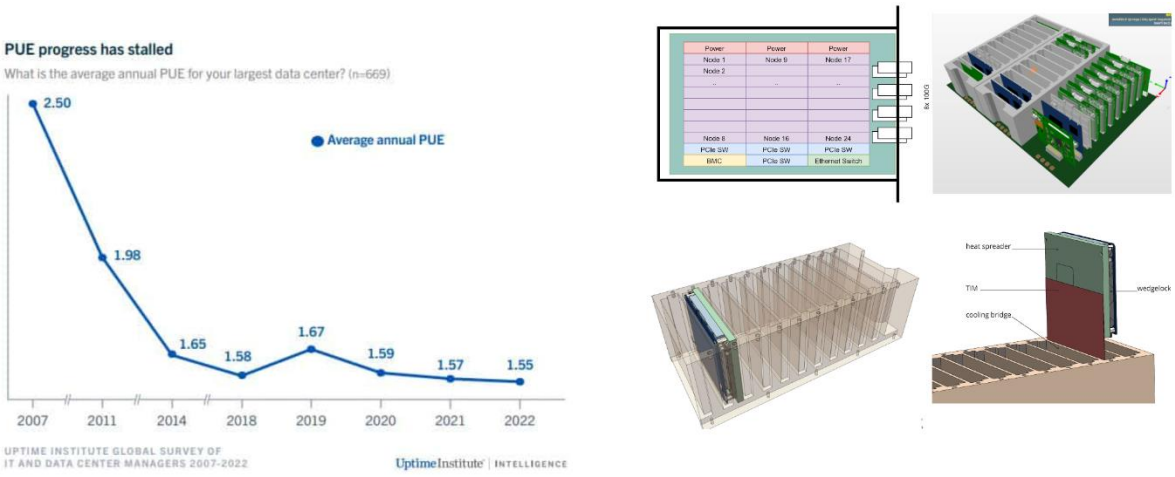

Figure 5. (a) PUE improvement rate across years, (b) HIRO EMDC design

The current EMDC design is based on the com-express form factor, which does not have PCIe gen5 on its roadmap. Currently HIRO is writing proposals for additional EU funding to develop a next generation based on the com-HPC form factor with PCIe gen 5 (including CXL capabilities).

3.2.2 SYCLOPS EMDC Overview

In the context of SYCLOPS, our EMDC deployment and experimental validation in WP5 will only focus on 8-node block. The envisioned hardware solution that will be deployed at the end of the project (EMDC V2.0 at M33) will have the following innovative properties:

1. Specifically designed monoblock chassis with innovative passive (i.e., fan less) two-phase cooling system based on Loop Thermosyphon (LTS) and nanoparticles. The EMDC monoblock is designed to be a modular system with 8 custom boards interchangeable (CPUs, GPUs, RISC-V FPGA board, etcetera), one power supply board, one ethernet switch board, and one PCIe switch board.

2. New backplane and connector including both dual x8 lane Gen5 PCIe interface and 4 x 10/25G Ethernet link for unprecedented intra- and inter-EMDC throughput. This will be one of the first small-form-factor EMDCs in the market to support PCIe Gen5.

3. New compact card design fitting the com-HPC form factor.

The EMDC will be heterogeneous in two ways. First, in terms of silicon type, we will implement energy efficient embedded Systems on a Chip (Intel, AMD, ARM) and the first FPGA system on chip (Xilinx Versal) and GPU system on chip (Nvidia Jetson Xavier). Second, in terms of cluster type, different nodes will support a varied selection of CPU, FPGA and GPU.

Multi-tier storage. The EMDC can support multi-tier storage.

- Tier 3: Each storage nodes carries 4x M2 NVMe node accessible via PCIe Gen 5.A storage centric EMDC block, 1 FPGA node + 7 Storage nodes (7x 8TB) enables in stream processing, big data ingestion, cleaning and storage.
- Tier 2: A mixed EMDC for training and inferencing: The tier for the raw data can be a storage centric EMDC with many storage nodes. EMDC has enough storage nodes to store the training data close to the accelerators.
- Tier 1: Storage on the compute and accelerator node, only accessible to the node itself. Used for storing a model, an AI agent, but also cleaned data to iteratively improve the model. Storage can also be used to store incoming data that needs to be processing/ inferencing by the agent.

The storage architecture can be customized to accommodate the growing datasets and data-variety, offering data storage federation across multiple data sources, chosen carefully to be the optimal placement for the corresponding different types of data; e.g. time-series data (often placed in graph-databases, other NoSQL databases), video, image, and audio files (often placed in general-purpose shared filesystem such as NFS), log data (often placed in Splunk, Hadoop), unstructured data (often placed in HDFS or S3, RDBMS, NoSQL, Business intelligence systems).

PCIe Gen5. PCIe 5.0 was introduced as a direct successor to PCIe 4.0 and is quickly becoming an popular alternative. It is both backward- and forward-compatible, meaning it will work with devices using past generations of PCIe as well as any future versions. It supports 400 gigabit Ethernet and can handle nearly double the throughput of PCIe 4.0, it allows for high-speed networking.

PCIe 5 brings huge boosts in performance speed and bandwidth. With Gen 1, we saw bandwidth speeds of 250MB/s and data transfer rates of 2.5 GT/s (gigatransfers) per lane. Now with Gen 5, with the PCIe next-gen rule, speeds will reach up to 32 GT/s of data transfer and 4GB/s of bandwidth per lane.

The additional bandwidth of PCIe 5.0 means that devices may be able to achieve the same throughput while using fewer lanes, thus freeing up the number of lanes available. For example, a graphics card that used to require x16 bandwidth to run may now run at the same speed with x8 lanes, thus freeing up x8 lanes for use. PCIe Gen 5 effectively allows these lanes to become more available for further additions via PCIe slots.

A dual fabric (PCIe, Ethernet) that supports high performing data ingestion and distribution and to provision sufficient bandwidth for any population of the 8 nodes (storage, acceleration, CPU, GPU centric or a mix) will be investigated. The PCIe switching will have two levels, a PCIe switch for each 8-node block and, if needed, an overarching PCIe switch to master three PCIe switches.

CXL: CXL, now at the 3.0 specification level, provides low-latency links and memory coherency between computing devices. It builds on the enormous momentum of PCI

Express (PCIe)technology by adopting the PCIe PHY as its physical interface. CXL 1.1/2.0 use the PCIe 5.0 PHY operating at 32 Gigatransfer per second (GT/s).

The CXL.memory protocol enables a host, such as a processor, to access device attached memory using load/store commands. Sharing memory resources between computing devices, such as a CPU host and an AI accelerator, can be enabled by using all three of the CXL protocols. For instance, a server with a CXL-connected accelerator would enable the accelerator to use the CPU's direct-attached memory for workloads which required greater memory capacity.

Currently the EMDC's work in AI-enabled federations. The recently started ACES (2023-2026) EU project will implement swarm technologies to improve the orchestration and scheduling of workloads across the heterogeneous autonomous EMDC's working in ad-hoc self-initiated federations. With current CXL technologies the swarm algorithms will be partially executed on one or more central locations. Once the CXL 3.0 standard is fully implemented and made available through hardware components the EMDC will be able to run the swarm technology fully distributed across the CXL infrastructure.

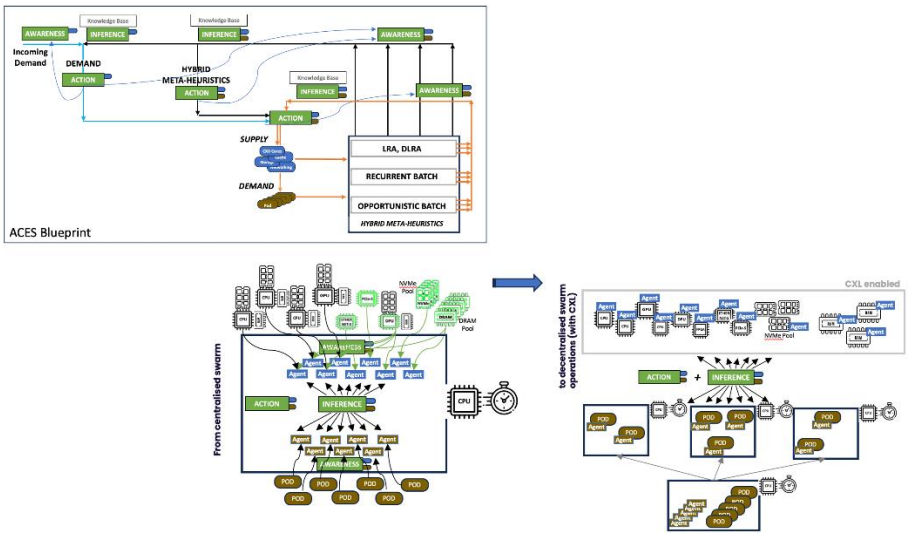

Figure 6. CXL and other technologies used for orchestration of workload in the EMDC

3.2.3 Interfaces & Integration

The EMDC outlined will be used as the central development environment in project SYCLOPS. The RISC-V bitstream described above will be integrated into the EMDC in addition to other processing hardware described in Section 3.2. Two versions of EMDC will be developed during the course of the project.

- **EDC v1.0 (M18)**: In order to facilitate early exploration of hardware prototypes developed in the project and to enable cross-layer compatibility checking and inter-layer integration, a first version of the edge data center (EDC) will be developed and deployed by M18. As building a miniaturized comHPC form factor of various boards will be done during the course of the project, EDC v1.0 will developed in standard form factors during COTS hardware with PCIe Gen5. We refer to this as EDC to distinguish it from the v2.0 EMDC.
- **EMDC v2.0 (M33)**: The second version of the data center will be the EMDC in comHPC form factor.

4 Platform Layer

Rise of SYCL. As the RISC-V ecosystem continues to evolve, we are starting to see both RISC-V acceleration solutions, where a host control CPU is used in tandem with a RISC-V accelerator, and even single ISA solution with a RISC-V host and RISC-V accelerator. This naturally necessitates an open programming model that supports a cross-vendor, cross-architecture computation offloading. Historically, the OpenCL standard has been instrumental in enabling hardware acceleration on a wide range of processors, including multi-vendor CPU, GPU, DSP, etcetera, for computer vision and HPC applications. OpenCL provided a "C"-like language for writing compute-intensive kernels that can be offloaded onto any supported accelerator using a runtime API. Thus, one possibility is to use OpenCL as the programming model for emerging RISC-V accelerators.

However, OpenCL has several challenges that makes it unsuitable as the model of choice for programming AI accelerators. First, the low-level nature of OpenCL was meant to directly expose data parallelism in underlying hardware while leaving everything else from data movement to kernel dispatch to developers leading to boilerplate code verbosity. Second, programs written in OpenCL are not single-source in nature as kernel code needs to be separated from host code, represented as strings and separately managed, complicating software development. Separate host and device source also loses strong type checking safety between host and device code. Third, while OpenCL provides code portability across processors, it does not provide support to guarantee performance portability as developers have to manually customize the code further to match properties of the underlying hardware. These issues with OpenCL are well known in the High-Performance Computing (HPC) community, where applications have evolved to adopt more general-purpose programming languages like C++ instead of C and FORTRAN, and HPC installations have expanded to adopt processors from more vendors. These challenges led to the development of custom HPC frameworks like RAJA and Kokkos that bridge the gap by providing C++ abstraction layers for portable parallel execution. Thus, in turn, spurred the development of SYCL, an open, industry-standard, programming model from Khronos group (who also maintain OpenCL).

SYCL is a higher-level programming model that brings hardware acceleration to mainstream C++ with the goal of making it easy to write new software similar to CUDA. Unlike OpenCL, SYCL provides a single-source programming model for both processors and accelerators, where kernel code and host code can co-exist in a single file, vastly simplifying development effort. Single-source programming not only enables strong type checking of host and device code, but more importantly, as shown in Figure 7, it enables a host of template libraries that broadens the code to support the latest patterns, and vastly expands the SYCL ecosystem to support data analysis, machine learning, and HPC.

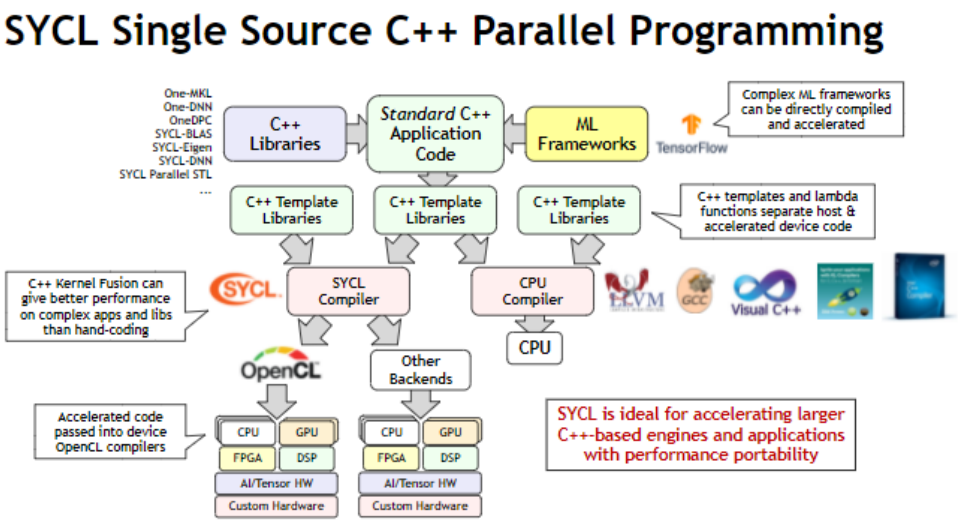

Figure 7: SYCL single-source parallel and heterogeneous programming framework

Despite recent advances, the SYCL ecosystem today lacks several tools and technologies that are required for it to be used as a drop-in substitute for proprietary solutions like CUDA: (i) there is no SYCL RISC-V compiler backend today that can compile a SYCL program to target RISC-V accelerators, (ii) there is no SYCL implementation today that supports a graph-based task runtime to enable efficient, cross-processor scheduling, (iii) there is no SYCL interpreter available today that can enable ad-hoc, interactive data analytics. These limitations create barriers for software developers and hinders the adoption of SYCL as the programming model of choice for AI acceleration.

In this section, we describe the components at the platform layer of SYCLOPS that will be developed to bridge this gap.

4.1 DPC++ Compiler and Runtime (CPLAY)

oneAPI is an open, cross-architecture programming model allowing developers to use a single codebase across multiple accelerator architectures such as GPUs and FPGAs. oneAPI uses SYCL at its core and this is used to implement a set of libraries including math and AI using the Data Parallel C++(DPC++)/C++ Compiler. DPC++ is a LLVM-based compiler project that implements compiler and runtime support for the SYCL language.

The existing DPC++ compiler already targets Intel, Nvidia and AMD GPUs. In the platform layer, we will enable a RISC-V/RVV backend for DPC++ and validate it on the RISC-V cores to support the execution of SYCL applications on RISC-V and other accelerators provided by the SYCLOPS infrastructure layer. We will rely on the oneAPI construction kit developed by CPLAY to make this possible.

oneAPI Construction Kit makes it possible to bring the components of oneAPI, in particular the DPC++ compiler, to new accelerator processor architectures. The following diagram shows how the oneAPI Construction Kit currently makes it possible to add new devices so that they can make use of the DPC++ SYCL compiler.

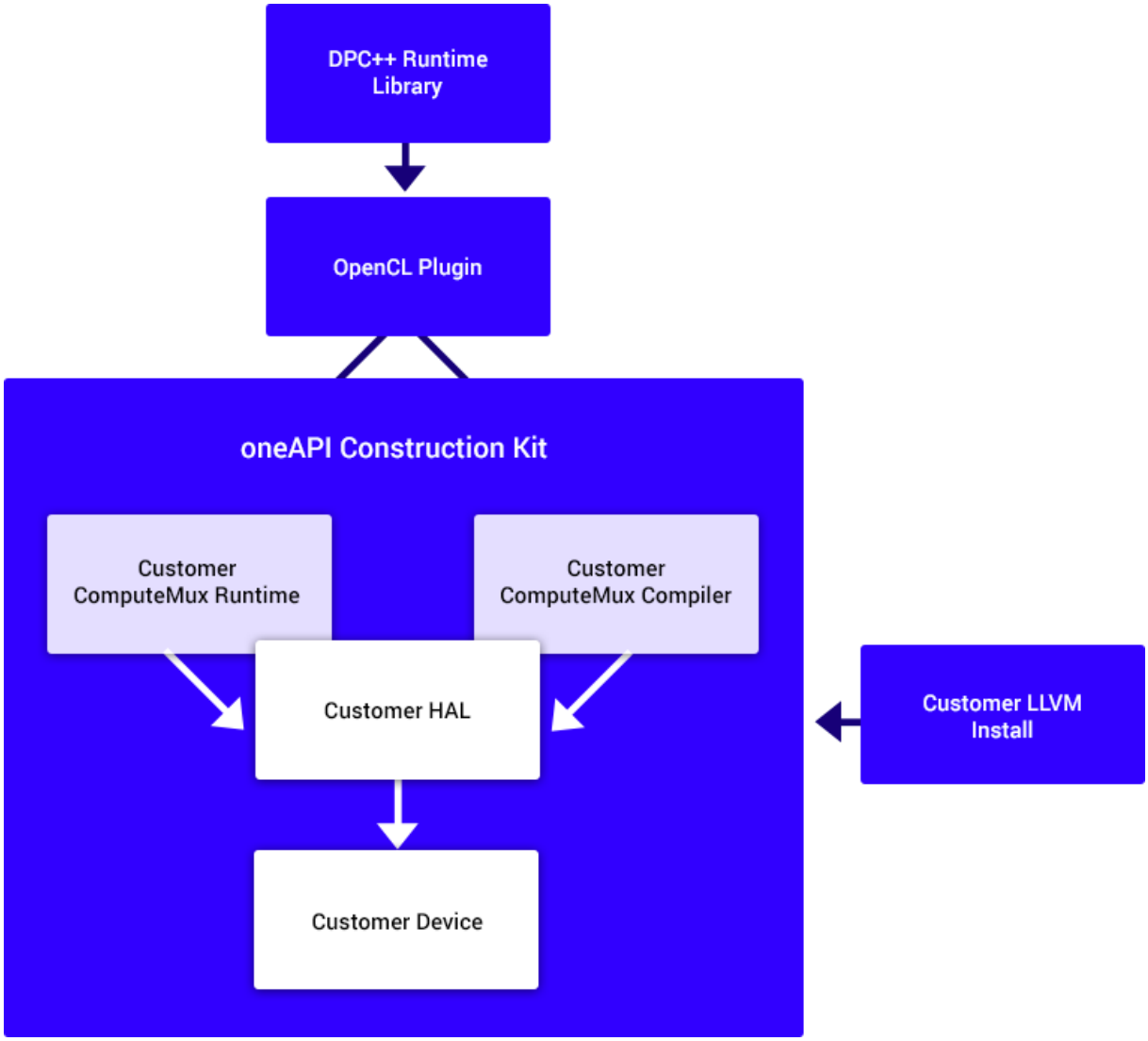

Figure 8: Components of the oneAPI construction kit

The DPC++ runtime allows OpenCL to be used as a plugin. The oneAPI ConstructionKit supports this interface and provides runtime and compiler modules. Support for new accelerators is done by developing a new custom target. A custom target is made up of three key parts:

- Runtime code (ComputeMux Runtime)
- Compiler code ComputeMux Compiler)
- An Optional HAL (Hardware Abstraction Layer) which gives static information and simplified runtime interfaces to a device to make getting started easier

The runtime code will run on the host device and will interface with the target device. It will handle aspects such as allocation of and reading/writing memory, queuing of commands and executing kernels on the device. The compiler code is typically based on a number of LLVM passes that will turn the original kernels into something matching the interfaces required to run a kernel on the device.

4.1.1 Interfaces and Integration

In SYCLOPS, the oneAPI construction kit will be used to support RISC-V accelerators. More specifically, the DPC++ compiler extended to support RISC-V. Further, the DPC++ compiler will also be fine-tuned to ensure scalable auto-vectorization and efficient utilization of vector engines in RVV cores. The DPC++ runtime will also be extended to support computational offloading to the RISC-V accelerator.

4.2 hipSYCL Compiler and Runtime (UHEI)

As part of the SYCLOPS project, UHEI will work on the hipSYCL implementation of SYCL, which is a portable, open-source SYCL implementation. Currently, hipSYCL supports executing kernels on CPUs as well as Intel, NVIDIA and AMD GPUs. Additionally, hipSYCL is designed to be flexible and supports many different compilation flows. This includes library-based compilation flows, compilation flows targeted at interoperability with existing proprietary models like CUDA as well as a standalone single-pass SYCL compiler.

In the context of SYCLOPS, we make two main contributions. First, we will generalize the hipSYCL compiler and runtime to support RISC-V. Second, we will add support for multi-device scheduling using a graph-based runtime. This work is intended to allow applications to seamlessly use multiple devices without explicit multi-device programming, and thereby simplifying the development process.

4.2.1 Interfaces and Integration

For RISC-V support, we will build on hipSYCL's generic single-pass compilation flow. In this compilation flow, hipSYCL compiles the input code in a single compiler pass to an intermediate representation (IR) that is based on the IR of the LLVM compiler infrastructure, and then relies on just-in-time compilation at runtime to lower this IR to target-specific formats. hipSYCL's generic single-pass compiler is designed to be extensible to new targets, and therefore is a suitable target for this work. However, since it was only introduced recently, some stabilization work will be required prior to being able to work on RISC-V offload.

From discussion with CPLAY, it was found that the ComputeAorta platform may potentially be leveraged by hipSYCL to generate code for RISC-V. This would allow UHEI and CPLAY to converge on a common lower layer when targeting RISC-V, which is attractive. In this case, it might be sufficient for hipSYCL to generate code in SPIR-V format (which it can already do) and add a new runtime backend that relies on ComputeAorta to submit SPIR-V to RISC-V devices. Should relying on ComputeAorta not be possible, hipSYCL's single-pass compiler will need to be extended with dedicated RISC-V code generation backends. Due to the resulting duplication of work in the SYCLOPS stack, this approach is not preferred.

The interfaces to the upper layers in the SYCLOPS stack are well defined by the SYCL and C++ specifications. Code from the application layer that conforms to these standards will be able to leverage our developments. For the multi- device scheduling however, it needs to be noted that SYCL in its current form was not designed for implicit multi-device execution and consequently, some extensions or restrictions to the SYCL API may be necessary. We will try to keep these changes as light as possible, and will provide documentation on the supported feature set in this execution mode.

Apart from this, the multi-device graph runtime is mostly self-contained, as it will be implemented entirely within the hipSYCL runtime library in a manner that is transparent for

other components. An important goal of graph runtimes is to improve utilization and hide kernel submission latencies. A first step therefore is to optimize latencies of the hipSYCL runtime in order to have a solid baseline for further developments. This requires some architectural changes of the current design, such as reducing the number of small object allocations e.g. by migrating to an object pool design.

The new multi-device scheduler can then be integrated into the existing architecture of the hipSYCL runtime library. Multiple angles of optimization are conceivable here, including performance modelling the device code, collecting execution statistics at runtime or leveraging our just-in-time compiler to modify code at runtime to perform kernel fusion or generate code specifically tailored to a particular device based on runtime knowledge.

Another interesting approach in addition to implementing such functionality on the SYCL level could also be to provide SYCL-accelerated higher-level algorithms such as C++ STL standard parallel algorithms, which then can be distributed across multiple devices. This has the advantage that the SYCL runtime then has more knowledge about the nature of the algorithm and communication patterns, and can potentially leverage this knowledge to provide smarter scheduling decisions.

4.3 SYCL Interpreter (CERN)

Exploratory or ad-hoc analytics is an interactive approach to data analytics where users iteratively search, analyze, and filter data to identify relevant information via rapid application development. Python and Jupyter notebooks are often the tools of choice for such analysis due to the ability to perform on-the-fly interpretation and Just-in-time compilation to support quick interactive development. Although SYCL is well suited for data-intensive application development given its potential to exploit cross-architecture parallelism, it is not yet a good fit for exploratory programming due to the long edit-compile-run cycles during development.

Our goal is to bridge this gap by building on Cling, a unique interactive C++ interpreter that has enabled interactive, notebook-based exploration of over 1EB of data in High-Energy Physics (HEP) applications at CERN. The Cling C++ interpreter is used to just-in-time compile and call runtime-generated C++ source code. This enables runtime type-discovery, for instance from data files, and type-safe, efficient, and optimized analysis in an exploratory, interactive mode. A typical interface for data explorative programming is Jupyter Notebooks; these notebooks see worldwide, wide-spread adoption in all data sciences. Cling is used by Jupyter's default C++ kernel. This work will allow any usage of Jupyter C++ kernels to also just-in-time compile ("interpret") SYCL code, through the addition of SYCL support to cling.

4.3.1 Interfaces and Integration

To this end, we expect that Cling will use OpenSYCL / hipSYCL as compiler backend. Cling will be provided with SYCL code, which gets compiled to LLVM intermediate representation (IR). In OpenSYCL's "single source, single compiler pass" (SSCP) mode, the compiler will then compile this IR code to the actual compute or accelerator backend. This single-pass compilation, together with the compilation of SYCL code to IR, will happen at runtime. OpenSYCL can compile SYCL code to CUDA; we will use this for benchmarking and comparisons.

5 Use Cases, Benchmarks & Libraries Layer

Having described the two core layers of SYCLOPS in Section 4, in this section, we provide a description of the three use cases together the relevant supporting libraries for each use case.

5.1 Autonomous systems & SYCL-DNN

PointNet [11] is a deep learning model designed for processing unordered point clouds, which are sets of 3D points representing objects or scenes. Pointnet's novel architecture can process raw point cloud data without needing preprocessing through voxelation or predefined grids. The model consists of two main modules: An encoder that uses shared multi-layer perceptron to map the input points independently to higher-dimensional space; max-pooling is then used to encode a global feature vector; these features are then reduced through fully-connected layers and combined with trainable weights and biases to calculate an affine transformation matrix which is then used to normalize the pose of the input points. A similar transformation is then done to the features produced from the pose normalised points to then generate a global feature vector describing the entire point cloud. The second module uses the global feature vector to perform either classification or segmentation through a series of MLPs.

PointNet and its extensions have a wide range of applications in the automotive, drone, and UAV (Unmanned Aerial Vehicles) industries due to their ability to process 3D point cloud data efficiently. These applications include object detection and recognition where PointNet can be used to detect and recognize identify obstacles, pedestrians, and vehicles in the environment by processing point clouds. Furthermore, PointNet can segment point clouds into meaningful parts to identify structures such as road surfaces, buildings and vegetation. Finally, its encoding module can be used as a preprocessing step to convert commonly used unstructured LiDAR/Radar data into meaningful features that can then be used for mapping, localization or path planning tasks.

We will be working on developing the point-net model using SYCL-DNN and oneDNN libraries. As existing libraries of primitives for accelerating neural networks, SYCL-DNN and oneDNN do not support operators or optimized variants crucial for autonomous systems use case. Thus, we will extend SYCL-DNN by implementing support specifically for PointNet. On the methodological front, a key challenge in deriving practical inference strategies for parameters of Deep/Convolution Neural Networks is that these models are often characterized by millions or sometimes billions of parameters. In this project we will implement the remaining operators required for the PointNet model such as *concat*. We will augment SYCL-DNN with an optimized implementation of this method and optimize all other operators to be able to use the RVV optimizations available when running on RISC-V cores.

5.2 High Energy Physics & SYCL-ROOT

In modern High Energy Physics (HEP) studies, a computational graph is defined as an abstraction model to express data flow, selection, and transformation as operators. Lorentz vector operations on input arrays are highly common in the construction of operators. Coordinate transformation (polar to and from Cartesian) and general linear algebra are examples. These basic blocks are found in almost every HEP analysis. These procedures are now implemented in C++ for CPU as part of ROOT, a specialized library of interconnected components that help scientists with everything from data storage and study

to display. In many cases, these actions are the primary computing factor, dominating CPU utilization, energy cost, and result latency.

For the HEP use case, ROOT's analysis interface RDataFrame will be used to implement an example analysis. The example analysis reads ROOT files, analyzes them statistically, and creates aggregation histograms of the results, corresponding to all the steps, end-to-end, of typical physics analyses. The same analysis will be implemented both in C++ and Python. The analysis computations can use ROOT's current non-SYCL functions. Some of the analysis computations will also be implemented as kernels in SYCL. These kernels will use Lorentz vector operations including coordinate transformations. The analysis using SYCL-kernels will be run both on CPU and GPU, using DPC++(ComputeCPP) and hipSYCL / OpenSYCL and its CUDA backend. The analysis without SYCL kernels will run only on the CPU.

The analysis will be run in multi-process mode (multiple analysis processes analyzing data in parallel on the same computer) and in multi-threaded mode (multiple threads analyzing data in parallel within the same process). Scheduling of GPU usage from concurrent analysis (multi-threaded and multi-process) will need to be studied and implemented. The different implementations and running modes of the example analysis will be benchmarked to compare their throughput. Throughput measures include physics particle collisions analyzed per second, gigabytes of uncompressed data processed per second, and overall CPU / GPU utilization. We also plan to provide energy measurements.

To guide development, benchmarking will use the Linux `perf` performance counter-based profiling on CPU in addition to other performance profiling and modelling tools that will be developed in SYCLOPS (described in Section 5.4). We expect to use Nsight Compute for code using the CUDA GPU backend. We will use flamegraphs to visualize the performance measurements and make them accessible to developers. The complete end-to-end workflows will also be benchmarked as indicated above (throughput, efficiency, and energy usage).

Among other features, ROOT provides a linear algebra computational library including Lorentz vectors. Relevant coordinate transformations and Lorentz vector operations will be re-implemented using SYCL as programming language. The ROOT build system will be enhanced to enable the inclusion of SYCL code, to be compiled with OpenSYCL / hipSYCL and / or ComputeCPP. This code can be invoked through C++ wrapper functions that offload large memory buffers to the device. Multiple user interface approaches will be studied in terms of performance and usability, where physicists can either program SYCL themselves, combining the operations from the Lorentz vector SYCL library, or where an abstraction allows computations to be combined and offloaded through generated SYCL code. Some of this code will likely be just-in-time compiled using the cling SYCL support.

5.3 Precision Oncology & SYCL-GAL

Precision oncology, or tailoring cancer treatment to a person's genetic make-up, has been heralded as the next frontier in oncology. Genetics is being utilized to guide therapeutic decision-making in breast cancer (BC), and the increasing use of biomarkers for customized cancer therapy is predicted to boost the precision medicine market to $119 billion by 2026. However, in order to make clinical adoption of precision medicine a reality, the biotechnology and healthcare industries must make fundamental advances on two fronts: intelligent analysis of heterogeneous datasets and scalable computation to derive timely, actionable, life-saving insights.

Because of significant improvements in genomic sequencing technology, it is now possible to sequence a full human genome for less than $1000 and receive genetic data on several molecular levels of the cell (genomic, transcriptome, protein expression, copy-number variations, and so on). However, in order to turn this genomic Big Data into useful insights for precision medicine, a systematic integrated study of a range of biological datasets is required. ACCELOM has created easy-to-deploy, reproducible multi-omics software pipelines that combine numerous molecular datasets from a host to investigate association patterns between molecular levels using unique, peer-reviewed statistical machine learning algorithms.

At a high level, our pipeline consists of the following steps.

1. Preprocessing: This step involves the preparation of the raw sequencing data for variant discovery. It includes quality control, trimming, alignment, sorting, marking duplicates, base quality score recalibration (BQSR), and indel realignment. These steps aim to remove low-quality or erroneous reads, improve the accuracy of base quality scores, and correct for alignment artefacts that may affect variant calling.

2. Variant calling: This step involves the identification of variants in the preprocessed sequencing data. It includes tools for detecting single nucleotide variants (SNVs), insertions and deletions (indels), copy number variants (CNVs), and structural variants (SVs). These tools use various statistical models and algorithms to distinguish true variants from sequencing errors or artefacts.

3. Variant annotation: This step involves the annotation of variants with functional and clinical information, such as gene names, protein domains, amino acid changes, mutation effects, pathway memberships, drug targets and clinical trials. Several databases and resources are available to provide these annotations, such as dbSNP, COSMIC, ClinVar, OncoKB and cBioPortal.

4. Variant filtering: This step involves the filtering of variants based on various criteria, such as quality scores, allele frequencies, functional effects, population frequencies, and clinical relevance. These filters aim to remove false positive variants or prioritize variants of interest for further analysis.

5. Variant evaluation: This step involves the evaluation of variants based on various metrics, such as sensitivity, specificity, precision, recall, F1 score, concordance, discordance, and Mendelian inheritance errors. These metrics aim to assess the accuracy and performance of the variant calling pipeline or compare different pipelines or datasets.

Among these steps, preliminary analysis revealed that the first two steps are the most time consuming. For these two steps, we rely on the well-established GATK pipeline. The time for each stage of the GATK pipeline can vary depending on several factors, such as the type and size of the sequencing data, the computational resources available, the choice of tools and parameters, and the complexity of the analysis. However, some rough estimates can be given based on previous studies and reports.

According to a white paper by Intel [9], the preprocessing step can take from 2 to 6 hours per sample for whole-genome sequencing data (30x coverage). The variant calling step can take from 2 to 4 hours per sample for whole-genome sequencing data. These two steps are the most time consuming in the entire workflow. A study by IBM confirms this finding [10], as they find that the preprocessing step can take from 3 to 9 hours per sample for whole-genome sequencing data (30x coverage) using a Power9 processor with 64 cores and 512

GB of memory. The variant calling step can take from 1 to 3 hours per sample using the same processor.

Given the time consuming nature of these two stages, hardware acceleration solutions have emerged to scale preprocessing and variant calling. Two popular solutions are Illumina Dragen, which relies on FPGAs to accelerate the GATK pipeline stages, and NVIDIA Parabricks, which instead uses NVIDIA GPUs. Both these solutions are closed source and proprietary. Through SYCLOPS, ACCELOM intends seeks to develop SYCL-GAL library and integrate it with its genomic data analysis pipeline.

The library will provide accelerated implementations of various steps of the GATK pipeline. To this end, we will first address sequence alignment, which can be seen as an extreme-scale clustering problem under the edit distance metric space. ACCELOM has recently developed Accel-Align, a new sequence aligner that uses randomized embedding algorithms to provide 10x speedup over state-of-the-art aligners on CPU [12]. The first objective will be to develop a SYCL-based version of Accel-Align, that can exploit accelerators. The next bottleneck after alignment is post-alignment data preparation. In this task, the aligned sequencing reads are subjected to a series of sorting, filtration, and quality estimation steps. These steps can be viewed as relational operations on a database. Thus, we will build on oneDB, a SYCL library of core data-parallel primitives that has been used to build portable, parallel, data analytics engines. The third bottleneck is a pair Hidden Markov Model (pairHMM) algorithm in variant calling (Haplotypecaller/Mutect2). The third objective is to develop a SYCL-based pairHMM algorithm. These three pieces of SYCL-GAL will be integrated to build a new version of the multi-omics pipeline.

Using publicly available datasets, and well established benchmarking methodologies, we will compare the SYCL-GAL-based pipeline with non-accelerated GATK and accelerated, but closed-soure, Parabricks and Dragen pipelines if possible. We will measure various aspects of the pipelines, including overall execution time, memory requirement, weak and strong scaling etcetera, and use it to compute various metrics like cost (in case of cloud-hosted pipeline) and energy consumption.

5.4 Performance profiling and Modeling

In order to develop the three libraries (SYCL-DNN, SYCL-ROOT, SYCL-GAL) and optimize their performance on a range of diverse processors, one needs performance profiling and performance modelling tools. Performance monitoring is a critical aspect of modern computational systems, providing detailed insight into the performance, which might be useful for driving software and hardware design optimizations. In this context, modern processor designs provide hardware performance counters to enable the capture of performance metrics over monitoring tools. The insight provided by configuring and polling the performance counters enables software developers to take full advantage of the available hardware resources, aiding in the optimization process. Similarly, hardware designers may leverage these tools to identify performance bottlenecks, a possible focus point during future design iterations.

Accessing hardware counters is not a trivial task since they are often inaccessible at the user's privilege level, where most applications operate. In this sense, it requires an interface in the operating system's kernel for configuring and reading the counters in an environment with an elevated privilege level (e.g., system call). Moreover, different architectures may offer

distinct sets of performance counters or use different names/indexes to access equivalent counters, posing a significant challenge when attempting to profile an application across multiple platforms. Thus, configuring and accessing these counters is also platform-dependent, requiring different sequences of instructions in each ISA.

To facilitate performance profiling across architectures, some tools standardize this process by mapping generic events to platform-specific counters and providing the means to access them, such as Perf, PAPI [1] and LIKWID [2]. Perf, a Linux kernel native tool, consists of a kernel module to access hardware counters and a command line tool, allowing for the performance analysis of applications through the terminal. It can produce reports of varying detail and complexity while annotating the source code or the binary with specific event counters, breaking down the performance profile by functions/kernels, and highlighting key sections of the application.

PAPI - Performance Application Programming Interface [1] is a portable interface for performance profiling, which allows the configuration of performance counters with both native and generic predefined events by mapping one or more platform-specific identifiers. LIKWID [2] provides both a command line application and a library for analyzing specific regions of an application, which provides native support for thread pinning and multicore profiling. Although its API may be less complex than PAPI's, the performance counters require separate configurations through a command line tool.

Although Perf, PAPI, and LIKWID provide powerful capabilities, some events that a processor’s hardware counters may not capture can be obtained through other methods. For instance, a user may wish to measure the number of retired instructions of a particular type for which a counter is unavailable. They can instead rely on binary instrumentation, which inserts instructions into the compiled binary that measure any number of events. Consequently, this process does not preserve the original performance, meaning instrumentation is better suited to characterizing the application’s structure and not its performance. For instance, Intel's Pin is a binary instrumentation tool, allowing for the dynamic just-in-time instrumentation of compiled binaries. Intel SDE, built atop Pin, is a tool that generates instruction traces and creates an opcode histogram, breaking down the number of executed instructions and grouping them into categories.

Simulation tools are a popular approach for determining an application’s instruction profile, which is a crucial part of performance modeling. While running an application in a simulated environment is slower and usually not representative of its real performance, it can help analyze it in greater detail, producing instruction traces and additional statistics to help build an opcode histogram. For example, the gem5 simulator [8] can simulate many different ISAs, also allowing simulation across different platforms. The computer architecture can also be simulated in greater detail, producing an extensive set of execution statistics along with the instruction trace.

5.4.1 Roofline Modeling

Roofline modeling is a well-established tool for performance analysis, particularly in the context of HPC. The performance is characterized by the ratio between achieved performance, usually expressed in terms of floating-point operations (FLOPS/s), and the arithmetic intensity of an application in terms of operations per byte (e.g., FLOPS/byte). By visually representing the performance of applications alongside the capabilities of the underlying architecture, the roofline model offers an intuitive understanding of the platform's capabilities and whether the application is exploiting them effectively. Furthermore, the

position of the application point in the roofline model offers additional insight into the possible performance bottleneck, which can help guide the software optimization process.

The Cache-Aware Roofline Model (CARM) [3] is a remarkably insightful performance model which considers the hierarchical nature of most modern memory subsystems, thereby providing additional insights when compared to the Original Roofline Model [4]. CARM provides a more detailed representation of the performance upper-bounds, depicting one roofline for each memory level, helping to accurately identify the performance bottlenecks of applications, with a special emphasis on data locality. In a nutshell, the roofline model has two regions: the memory-bound and the compute-bound region. The former focuses the performance analysis on the respective memory level's bandwidth and displaying low arithmetic intensity. The core's computational throughput instead limits compute-bound applications and have a high arithmetic intensity. The two zones join at the ridge point, where applications fully exercise both processor subsystems simultaneously.

In this regard, CARM's construction requires two benchmarks to be integrated: the memory and the compute benchmark. The memory benchmark measures the sustainable bandwidth of each memory level and consists of a set of microbenchmarks, each operating on an array of a different size, in order to allow for targeting different memory levels. Each microbenchmark traverses the previously referred array, performing a load or store operation on each element, the ratio of which can be adjusted to reach the peak bandwidth. The compute benchmark measures the peak compute performance of the architecture, e.g., the amount of delivered floating-point operations per unit of time, using instructions that maximize the FLOPs per cycle, such as SIMD instructions performing fused multiply and add.

This type of performance modeling has been integrated into popular x86 CPU performance analysis tools such as the Intel Advisor or AMD's μProf. Roofline modeling capabilities have also been added to performance analysis tools for GPUs, such as AMD's Omniperf and NVIDIA Nsight.

5.4.2 Performance Profiling and Modeling in RISC-V Systems

Similarly to other ISAs, different RISC-V architectures have different sets of hardware performance counters available. However, the standard mandates three counters that must be present across all architectures: clock cycles, instructions retired, and time counters. While these are sufficient for a simple performance analysis based on execution time, more detailed profiling requires the configuration and availability of additional performance counters. Tools such as Perf or PAPI would enable users to profile their applications in a portable manner, not requiring detailed knowledge of the ISA (to configure and read the counters) or the architecture (to know which events are counted and their identifiers).

The authors in [5] propose a RISC-V performance profiling framework, extending the limited support for RISC-V ISA performance monitoring in Perf within the associated kernel module, PAPI, and libpfm4. Additionally, it lists the events of two RISC-V architectures in both Perf and PAPI, mapping them to PAPI's preset events.

In this regard, the RISC-V specification still needs to provide a standardized way to obtain power and energy measurements. However, custom implementations are free to add interfaces for power consumption measurement whereas avoiding conflict with the base ISA standards. Such an interface, which could operate similarly to Intel's RAPL, would be essential in developing insightful models considering the energy efficiency of applications and architectures [6].

In order to construct roofline models in RISC-V systems, it is essential first to determine the system's peak arithmetic performance and memory bandwidth. While a more robust set of performance counters may be necessary to profile an application in the roofline's context effectively, the three base counters in the RISC-V specification are sufficient to build the model. Considering the programmatically generated micro-benchmarks, the number of floating-point operations and bytes can be determined during generation, while the cycle counter provides the execution time. Regarding the target architecture implementing the RISC-V Vector extension, the previously elaborated benchmarks can be adjusted to use vector instructions instead. Moreover, the benchmarking process may benefit from a cycle counter in the accelerator for improved accuracy for a decoupled vector architecture. However, it can fall back on parent core's counters and existing synchronization methods.

In order to improve the accuracy of the benchmarking process, some constraints must be taken into account to provide consistent results. A significant contributor to the inconsistency of the measurements is the operating system's influence, which can be minimized by pinning threads to specific cores and increasing their scheduling priority. Other external factors can be mitigated by both normalizing the execution time of each microbenchmark and running them a large number of times, eliminating outliers.

In the context of RISC-V, the ISA simulator can generate the execution's instruction trace, which can be used to build the application's instruction profile. The work proposed in [7] develops this concept, thus providing a function-level breakdown of an application's instruction profile through Spike simulation and binary analysis, using it as a base for timing and energy models. These features can be potentially packaged in a dedicated tool, which can handle benchmark generation, execution and performance measurement, and model construction. Automating this process means the CARM can be used as a tool for comparative evaluation, helping to quickly determine the capabilities of various designs, representing them intuitively, and facilitating their comparison.

Furthermore, an automated tool may enable performance modeling to be used in the hardware design process. While performance models have traditionally been applied to software optimization, identifying performance bottlenecks and possible optimization approaches, whether those same insights can help guide the hardware design process should be investigated. A configurable architecture could be scaled according to the identified bottleneck of a particular application or set of applications, improving performance for the target use cases.

5.5 Interfaces and Integration

All three libraries will rely on the SYCL compilers from the platform layer described in Section 4 to compile code to relevant processor backends. They will all be deployed on the EMDC described in Section 3 and evaluated against a variety of multi-vendor accelerators. The profiling and modelling tools described in Section 5.4 will be used to optimize the performance of these libraries. Employing performance modeling techniques for the three SYCLOPS use-case applications will require the evaluation of their instruction profile and determining their arithmetic intensity in the context of roofline modeling. Three possible approaches to this problem are implementing the necessary hardware counters, the code or binaries' instrumentation, or the instruction trace's simulation and analysis. The resulting insights will help drive optimization efforts, identifying whether the application effectively uses the available resources and which optimization approach should be taken. Finally, an in-depth benchmarking study will be performed using each library in the context of its use case to identify improvements achieved via SYCLOPS.

6 References

[1] V. M. Weaver et al., “Measuring Energy and Power with PAPI,” in 2012 41st International Conference on Parallel Processing Workshops, Pittsburgh, PA, USA: IEEE, Sep. 2012, pp. 262–268. doi: 10.1109/ICPPW.2012.39.

[2] J. Treibig, G. Hager, and G. Wellein, “LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments.” arXiv, Jun. 30, 2010. Accessed: Jul. 26, 2023. [Online]. Available: http://arxiv.org/abs/1004.4431

[3] A. Ilic, F. Pratas, and L. Sousa, “Cache-aware Roofline model: Upgrading the loft,” IEEE Comput. Arch. Lett., vol. 13, no. 1, pp. 21–24, Jan. 2014, doi: 10.1109/L-CA.2013.6.

[4] S. Williams, A. Waterman, and D. Patterson, “Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures,” 1407078, Sep. 2009. doi: 10.2172/1407078.

[5] J. M. Domingos, T. Rocha, and N. Neves, “Supporting RISC-V Performance Counters Through Linux Performance Analysis Tools”.

[6] A. Ilic, F. Pratas, and L. Sousa, “Beyond the Roofline: Cache-Aware Power and Energy-Efficiency Modeling for Multi-Cores,” IEEE Trans. Comput., vol. 66, no. 1, pp. 52–58, Jan. 2017, doi: 10.1109/TC.2016.2582151.

[7] J. Silveira et al., “Prof5: A RISC-V profiler tool,” in 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Bordeaux, France: IEEE, Nov. 2022, pp. 201–210. doi: 10.1109/SBAC-PAD55451.2022.00031.

[8] J. Lowe-Power et al., “The gem5 Simulator: Version 20.0+,” arXiv:2007.03152 [cs], Sep. 2020, Accessed: Dec. 28, 2021. [Online]. Available: http://arxiv.org/abs/2007.03152.

[9] Intel, Deploying GATK Best Practices, https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/deploying-gatk-best-practices-paper.pdf

[10] IBM, A guide to GATK best practice pipeline performance and optimization on the IBM OpenPOWER system, https://www.ibm.com/downloads/cas/ZJQD0QAL

[11] Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas, PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, https://arxiv.org/abs/1612.00593

[12] Yan Y., Chaturvedi N., Appuswamy R., Accel-Align: a fast sequence mapper and aligner based on the seed–embed–extend method, BMC Bioinformatics 2021